Hadoop2 basic and Rhipe package

Xiaosu Tong

Department of Statistics
Purdue University

March 10, 2016

Hadoop

- Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features and of the MapReduce computing paradigm
- Hadoops HDFS is a highly fault-tolerant distributed file system. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance
- THe block size and replication factor are configurable.
- MapReduce is a programming model and an associated implementation for processing and generating key-value pairs on HDFS.

Hadoop

- Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features and of the MapReduce computing paradigm
- Hadoops HDFS is a highly fault-tolerant distributed file system. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance
- THe block size and replication factor are configurable.
- MapReduce is a programming model and an associated implementation for processing and generating key-value pairs on HDFS.

Hadoop

- Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features and of the MapReduce computing paradigm
- Hadoops HDFS is a highly fault-tolerant distributed file system. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance
- THe block size and replication factor are configurable.
- MapReduce is a programming model and an associated implementation for processing and generating key-value pairs on HDFS.

Hadoop

- Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features and of the MapReduce computing paradigm
- Hadoops HDFS is a highly fault-tolerant distributed file system. It stores each file as a sequence of blocks. The blocks of a file are replicated for fault tolerance
- THe block size and replication factor are configurable.
- MapReduce is a programming model and an associated implementation for processing and generating key-value pairs on HDFS.

Hadoop: Data Storage

- namenode(master) - Manages the directory tree of the Hadoop File System (HDFS), it holds the meta data for the HDFS. When in use, all this information is stored in main memory, but also stored in disk.
    - fsimage - Its the snapshot of the filesystem when namenode started
    - Edit logs - Its the sequence of changes made to the filesystem after namenode started
- secondarynamenode - Offloads HDFS checkpoint support for the namenode. It is not a namenode failover or backup as the name may imply.
    - It gets the edit logs from the namenode in regular intervals and applies to fsimage
    - Once it has new fsimage, it copies back to namenode
- datanode(worker) - Stores data on the local drives of nodes as part of the distributed HDFS. They store and retrieve blocks when they are told to.

Hadoop: Data Storage

- namenode(master) - Manages the directory tree of the Hadoop File System (HDFS), it holds the meta data for the HDFS. When in use, all this information is stored in main memory, but also stored in disk.
  - fsimage - Its the snapshot of the filesystem when namenode started
  - Edit logs - Its the sequence of changes made to the filesystem after namenode started
- secondarynamenode - Offloads HDFS checkpoint support for the namenode. It is not a namenode failover or backup as the name may imply.
  - It gets the edit logs from the namenode in regular intervals and applies to fsimage
  - Once it has new fsimage, it copies back to namenode
- datanode(worker) - Stores data on the local drives of nodes as part of the distributed HDFS. They store and retrieve blocks when they are told to.

Hadoop: Data Storage

- namenode(master) - Manages the directory tree of the Hadoop File System (HDFS), it holds the meta data for the HDFS. When in use, all this information is stored in main memory, but also stored in disk.
    - fsimage - Its the snapshot of the filesystem when namenode started
    - Edit logs - Its the sequence of changes made to the filesystem after namenode started
- secondarynamenode - Offloads HDFS checkpoint support for the namenode. It is not a namenode failover or backup as the name may imply.
    - It gets the edit logs from the namenode in regular intervals and applies to fsimage
    - Once it has new fsimage, it copies back to namenode
- datanode(worker) - Stores data on the local drives of nodes as part of the distributed HDFS. They store and retrieve blocks when they are told to.

Hadoop: Data Storage

- namenode(master) - Manages the directory tree of the Hadoop File System (HDFS), it holds the meta data for the HDFS. When in use, all this information is stored in main memory, but also stored in disk.
  - fsimage - Its the snapshot of the filesystem when namenode started
  - Edit logs - Its the sequence of changes made to the filesystem after namenode started
- secondarynamenode - Offloads HDFS checkpoint support for the namenode. It is not a namenode failover or backup as the name may imply.
  - It gets the edit logs from the namenode in regular intervals and applies to fsimage
  - Once it has new fsimage, it copies back to namenode
- datanode(worker) - Stores data on the local drives of nodes as part of the distributed HDFS. They store and retrieve blocks when they are told to.

Hadoop: Data Storage

- namenode(master) - Manages the directory tree of the Hadoop File System (HDFS), it holds the meta data for the HDFS. When in use, all this information is stored in main memory, but also stored in disk.
    - fsimage - Its the snapshot of the filesystem when namenode started
    - Edit logs - Its the sequence of changes made to the filesystem after namenode started
- secondarynamenode - Offloads HDFS checkpoint support for the namenode. It is not a namenode failover or backup as the name may imply.
    - It gets the edit logs from the namenode in regular intervals and applies to fsimage
    - Once it has new fsimage, it copies back to namenode
- datanode(worker) - Stores data on the local drives of nodes as part of the distributed HDFS. They store and retrieve blocks when they are told to.

Hadoop: Data Storage

- namenode(master) - Manages the directory tree of the Hadoop File System (HDFS), it holds the meta data for the HDFS. When in use, all this information is stored in main memory, but also stored in disk.
  - fsimage - Its the snapshot of the filesystem when namenode started
  - Edit logs - Its the sequence of changes made to the filesystem after namenode started
- secondarynamenode - Offloads HDFS checkpoint support for the namenode. It is not a namenode failover or backup as the name may imply.
  - It gets the edit logs from the namenode in regular intervals and applies to fsimage
  - Once it has new fsimage, it copies back to namenode
- datanode(worker) - Stores data on the local drives of nodes as part of the distributed HDFS. They store and retrieve blocks when they are told to.

Hadoop: Data Storage

- namenode(master) - Manages the directory tree of the Hadoop File System (HDFS), it holds the meta data for the HDFS. When in use, all this information is stored in main memory, but also stored in disk.
  - fsimage - Its the snapshot of the filesystem when namenode started
  - Edit logs - Its the sequence of changes made to the filesystem after namenode started
- secondarynamenode - Offloads HDFS checkpoint support for the namenode. It is not a namenode failover or backup as the name may imply.
  - It gets the edit logs from the namenode in regular intervals and applies to fsimage
  - Once it has new fsimage, it copies back to namenode
- datanode(worker) - Stores data on the local drives of nodes as part of the distributed HDFS. They store and retrieve blocks when they are told to.

Hadoop: Computation

Hadoop1

- jobtracker - Schedules and issues map reduce jobs for tasktracker nodes across the cluster.
- tasktracker - Executes the map and reduce jobs issued by the jobtracker.

Hadoop: Computation

Hadoop1

- jobtracker - Schedules and issues map reduce jobs for tasktracker nodes across the cluster.
- tasktracker - Executes the map and reduce jobs issued by the jobtracker.

Hadoop2: Computation

Hadoop2

Resource Manager(RM) node:

- job is submitted to Resource Manager (asks for job ID, checks the output path ...)
- Applications Manager(AsM) - manages running jobs in the cluster.
- Scheduler - manages and enforces the resource scheduling policy in the cluster

NodeManager(NM) node:

- ApplicationMaster(AM) - A per-job master that manages the application's life cycle jobs on the cluster.
- Container, it is an Unix process which is assigned with specific amount of core and memory.

Hadoop2: Computation

Hadoop2

Resource Manager(RM) node:

- job is submitted to Resource Manager (asks for job ID, checks the output path ...)
- Applications Manager(AsM) - manages running jobs in the cluster.
- Scheduler - manages and enforces the resource scheduling policy in the cluster

NodeManager(NM) node:

- ApplicationMaster(AM) - A per-job master that manages the application's life cycle jobs on the cluster.
- Container, it is an Unix process which is assigned with specific amount of core and memory.

Hadoop2: Computation

Hadoop2

Resource Manager(RM) node:

- job is submitted to Resource Manager (asks for job ID, checks the output path ...)
- Applications Manager(AsM) - manages running jobs in the cluster.
- Scheduler - manages and enforces the resource scheduling policy in the cluster

NodeManager(NM) node:

- ApplicationMaster(AM) - A per-job master that manages the application's life cycle jobs on the cluster.
- Container, it is an Unix process which is assigned with specific amount of core and memory.

Hadoop2: Computation

Hadoop2

Resource Manager(RM) node:

- job is submitted to Resource Manager (asks for job ID, checks the output path ...)
- Applications Manager(AsM) - manages running jobs in the cluster.
- Scheduler - manages and enforces the resource scheduling policy in the cluster

NodeManager(NM) node:

- ApplicationMaster(AM) - A per-job master that manages the application's life cycle jobs on the cluster.
- Container, it is an Unix process which is assigned with specific amount of core and memory.

Hadoop2: Computation

Hadoop2

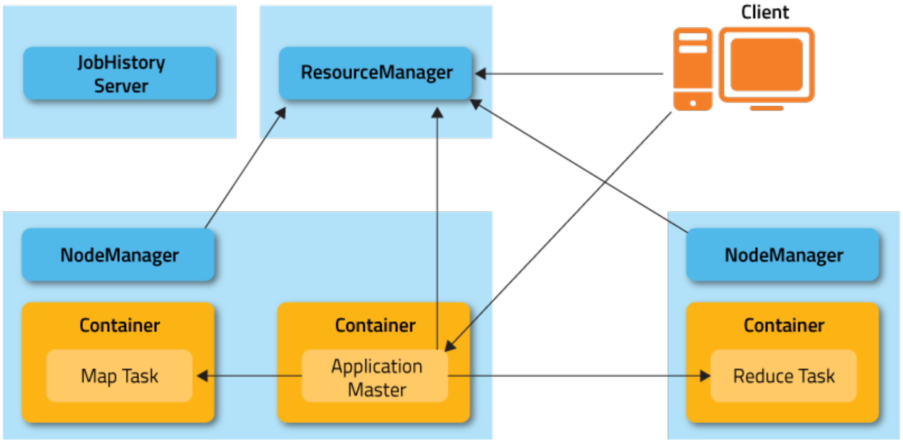
Resource Manager(RM) node:

- job is submitted to Resource Manager (asks for job ID, checks the output path ...)
- Applications Manager(AsM) - manages running jobs in the cluster.
- Scheduler - manages and enforces the resource scheduling policy in the cluster

NodeManager(NM) node:

- ApplicationMaster(AM) - A per-job master that manages the application's life cycle jobs on the cluster.
- Container, it is an Unix process which is assigned with specific amount of core and memory.

Hadoop2: Computation



Hadoop2: Computation

- Client $\leftrightarrow$ ApplicationsManager in ResourceManager
- ResourceManager(RM) $\leftrightarrow$ NodeManager. Finds an available container for running the ApplicationMaster
- ApplicationMaster $\leftrightarrow$ ResourceManager. Ask for containers for all map and reduce tasks.
- ApplicationMaster $\leftrightarrow$ NodeManager. Starts the containers and run JVMs.

Hadoop2: Computation

- Client $\leftrightarrow$ ApplicationsManager in ResourceManager
- ResourceManager(RM) $\leftrightarrow$ NodeManager. Finds an available container for running the ApplicationMaster
- ApplicationMaster $\leftrightarrow$ ResourceManager. Ask for containers for all map and reduce tasks.
- ApplicationMaster $\leftrightarrow$ NodeManager. Starts the containers and run JVMs.

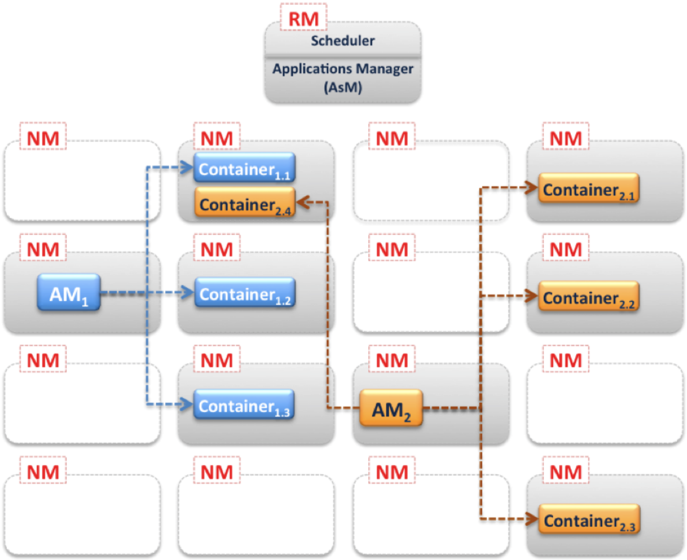
Hadoop2: Computation

- Client $\leftrightarrow$ ApplicationsManager in ResourceManager
- ResourceManager(RM) $\leftrightarrow$ NodeManager. Finds an available container for running the ApplicationMaster
- ApplicationMaster $\leftrightarrow$ ResourceManager. Ask for containers for all map and reduce tasks.
- ApplicationMaster $\leftrightarrow$ NodeManager. Starts the containers and run JVMs.

Hadoop2: Computation

- Client $\leftrightarrow$ ApplicationsManager in ResourceManager
- ResourceManager(RM) $\leftrightarrow$ NodeManager. Finds an available container for running the ApplicationMaster
- ApplicationMaster $\leftrightarrow$ ResourceManager. Ask for containers for all map and reduce tasks.
- ApplicationMaster $\leftrightarrow$ NodeManager. Starts the containers and run JVMs.

Hadoop2: Computation



MapReduce

- Map function: mapping one input key-value pair to one or multiple intermediate key-value pairs
- Reduce function: aggregating multiple key-value pairs who shares same key to be one key-value pair.
- Shuffle and Sort

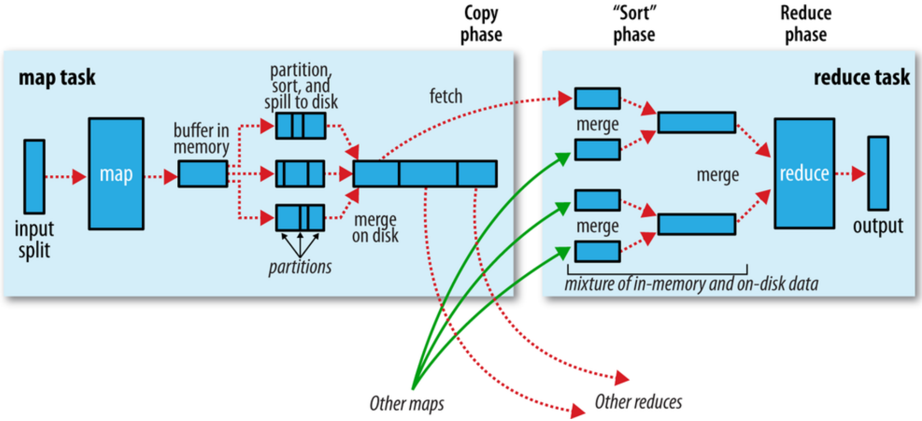
MapReduce

- Map function: mapping one input key-value pair to one or multiple intermediate key-value pairs
- Reduce function: aggregating multiple key-value pairs who shares same key to be one key-value pair.
- Shuffle and Sort

MapReduce

- Map function: mapping one input key-value pair to one or multiple intermediate key-value pairs
- Reduce function: aggregating multiple key-value pairs who shares same key to be one key-value pair.
- Shuffle and Sort

MapReduce


Copy phase
"Sort" phase
Reduce phase
map task
reduce task
input split
map
buffer in memory
partition, sort, and spill to disk
partitions
merge on disk
fetch
merge
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces

Rhipe

RHIPE

- It was first developed by Saptarshi Guha as part of his PhD thesis in the Purdue Statistics Department.
- RHIPE (hree-pay') is the R and Hadoop Integrated Programming Environment. RHIPE is a merger of R and Hadoop. It is a R package that communicates with Hadoop to carry out the big, parallel computations.

Rhipe

RHIPE

- It was first developed by Saptarshi Guha as part of his PhD thesis in the Purdue Statistics Department.
- RHIPE (hree-pay') is the R and Hadoop Integrated Programming Environment. RHIPE is a merger of R and Hadoop. It is a R package that communicates with Hadoop to carry out the big, parallel computations.

Prerequisites for Rhipe

- Protocol Buffers 2.5

  Protocol Buffers are a method of serializing structured data. They are useful in developing programs to communicate with each other over a wire or for storing data.

  ```
  tar -xzf protobuf-2.5.tar.gz
  cd protobuf-2.5
  sudo ./configure
  sudo make
  sudo make install
  ```

  Protocol Buffers library will be located in `/usr/local/lib`

  ```
  sudo ln -s protobuf-2.5/lib/lib* /usr/local/lib/
  ```

- rJava library
- testthat

Rhipe Installation

- environment setting in `.bashrc`

```
export PKG_CONFIG_PATH=
/usr/local/lib/pkgconfig/
export LD_LIBRARY_PATH=
/usr/local/lib:
/usr/lib/R/lib:
/usr/lib/jvm/java-6-openjdk-amd64/jre/lib/amd64/server
```

Rhipe Installation

- Download Rhipe pacakge

```
wget http://ml.stat.purdue.edu/rhipebin/
Rhipe_0.75.1.6.tar.gz
```

- Install Rhipe package

```
R CMD INSTALL Rhipe_0.75.1.6.tar.gz

>install.packages(
+"Rhipe_0.75.1.6.tar.gz", repos=NULL, type="source"
+)
```

Pushing

- Every node in the cluster should have exactly same R and all installed packages including Rhipe.

```
> library(R)
> rhinit()
> hdfs.setwd("/app/hadoop")
> bashRhipeArchive("RhipeLib")
```

  This step is only needed to be done for one time.

Start Rhipe

```
> library(Rhipe)
> rhinit()
> rhoptions(zips = "/app/hadoop/RhipeLib.tar.gz")
> rhoptions(
+  runner =
+   "sh ./RhipeLib/library/Rhipe/bin/RhipeMapReduce.sh"
+)
```

First Rhipe Job: words counts

- map expression
- reduce expression
- excution function

First Rhipe Job: words counts

- input key-value pairs:

```
(1:"A singular fatality has ... of every description.")
(2:"Vasari says, and rightly, ... pages of Manuscript.")
...
(30:"Alexander von Humboldt ... of Leonardo's genius:")
```

- map step
- output key-value pairs:

```
("A":1), ("singular",1), ("fatality":1), ("has":1), ...
("of":1), ("Leonardos":1), ("genius":1)
```

First Rhipe Job: words counts

- map expression:

```
map <- expression({
  for(i in seq_along(map.keys)) {
    line = gsub("[[:punct:]]", "", map.values[[i]])
    line = strsplit(line, split=" +")[[1]]
    for(word in line) {
      rhcollect(word, 1)
    }
  }
})
```

`map.keys` and `map.values` are two list objects in R which created by Rhipe. They are all the input keys and input values correspondingly.

First Rhipe Job: words counts

- input file is text, map expression will be evaluated on each block.
- `for` will loop over all rows in that block.
- `rhcollect` function will collect intermediate key-value pairs and write them onto local disk, not HDFS yet.

First Rhipe Job: words counts

- input file is text, map expression will be evaluated on each block.
- `for` will loop over all rows in that block.
- `rhcollect` function will collect intermediate key-value pairs and write them onto local disk, not HDFS yet.

First Rhipe Job: words counts

- input file is text, map expression will be evaluated on each block.
- `for` will loop over all rows in that block.
- `rhcollect` function will collect intermediate key-value pairs and write them onto local disk, not HDFS yet.

First Rhipe Job: words counts

- Between the map phase and reduce phase of a MapReduce job, Hadoop sends all the intermediate values for a given key to the reducer.
- The intermediate values for a given key are located on several compute nodes and need to be shuffled (sent across the network).
- Some operations do not need access to all of the data (intermediate values).
- Combiner is that the reduce run locally on mapper outputs before they are sent for the final reduce.

First Rhipe Job: words counts

- Between the map phase and reduce phase of a MapReduce job, Hadoop sends all the intermediate values for a given key to the reducer.
- The intermediate values for a given key are located on several compute nodes and need to be shuffled (sent across the network).
- Some operations do not need access to all of the data (intermediate values).
- Combiner is that the reduce run locally on mapper outputs before they are sent for the final reduce.

First Rhipe Job: words counts

- Between the map phase and reduce phase of a MapReduce job, Hadoop sends all the intermediate values for a given key to the reducer.
- The intermediate values for a given key are located on several compute nodes and need to be shuffled (sent across the network).
- Some operations do not need access to all of the data (intermediate values).
- Combiner is that the reduce run locally on mapper outputs before they are sent for the final reduce.

First Rhipe Job: words counts

- Between the map phase and reduce phase of a MapReduce job, Hadoop sends all the intermediate values for a given key to the reducer.
- The intermediate values for a given key are located on several compute nodes and need to be shuffled (sent across the network).
- Some operations do not need access to all of the data (intermediate values).
- Combiner is that the reduce run locally on mapper outputs before they are sent for the final reduce.

First Rhipe Job: words counts

- input key-value pairs:

```
("A":1), ("A":1), ..., ("A":1), ("singular",1), ...,
("singular",1), ("fatality":1), ("has":1), ..., ("has":1),
("of":1), ..., ("of":1), ("Leonardos":1), ("genius":1)
```

- reduce step
- output key-value pairs:

```
("A":35), ("singular",4), ("fatality":1), ("has":10), ...
("of":33), ("Leonardos":1), ("genius":2)
```

First Rhipe Job: words counts

- reduce expression:

```
reduce <- expression(
  pre = {
    count = 0
  },
  reduce = {
    count = count + sum(unlist(reduce.values))
  },
  post = {
    rhcollect(reduce.key, count)
  }
)
```

`reduce.key` is one of the unique output key from map step. `reduce.values` is a list object in R which collects all values corresponding to the `reduce.key`.

First Rhipe Job: words counts

- excution function:

```
mr <- rhwatch(
  map      = map,
  reduce   = reduce,
  input    = rhfmt(
    "/app/hadoop/words.txt",type="text"
  ),
  output   = rhfmt(
    "/app/hadoop/wordcount", type="sequence"
  ),
  mapred   = list( mapred.reduce.tasks=5 ),
  readback = FALSE,
  combiner = TRUE
)
```

First Rhipe Job: words counts

- read the result back to R

  ```
  rst <- rhread("/app/hadoop/wordcount")
  ```
- key-value pairs will be read back as a list object in R.
- Each element is a list with length two. First is the key, and the second is the value.

Download in Parallel

Huge number of data files to be downloaded
Airline dataset

`http://stat-computing.org/dataexpo/2009/1987.csv.bz2`

Download in Parallel

```
map <- expression({
  lapply(map.values, function(r){
    x = 1986 + r
    on <- sprintf(
      "http://stat-computing.org/dataexpo/2009/%s.csv.bz2",
       x
    )
    fn <- sprintf("./tmp/%s.csv.bz2", x)
    system(sprintf("wget  %s --directory-prefix ./tmp", on))
    system(sprintf("bunzip2 %s", fn))
  })
})
```

Download in Parallel

```
mr <- rhwatch(
  map      = map,
  input    = rep(length(1987:2008), 2),
  output   = rhfmt("/app/hadoop/dowload", type="sequence"),
  mapred   = list( mapred.reduce.tasks=0),
  readback = FALSE
)
```